ESD-TR-69-371
ESTI file copy

MTR-743

# THE MI-3 ASSEMBLER REFERENCE MANUAL

R. W. Cornelli

DECEMBER 1969



Prepared for

DIRECTORATE OF PLANNING AND TECHNOLOGY
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts




Project 700A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-68-C-0365

ESD-TR-69-371

MTR-743

# THE MI-3 ASSEMBLER REFERENCE MANUAL

R. W. Cornelli

DECEMBER 1969

Prepared for

DIRECTORATE OF PLANNING AND TECHNOLOGY
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts




Project 700A
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-68-C-0365

## FOREWORD

This report describes a one pass assembler built by the MITRE Corporation for a family of microprogrammable computers. It is in partial fulfillment of Project 7120 under Contract No. F19(628)-68-C-0365. It was prepared under the cognizance of Mr. Robert W. Cornelli of the MITRE Corporation, Bedford, Massachusetts. The USAF project monitor is Mr. Russell A. Meier.

## REVIEW AND APPROVAL

Publication of this technical report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

WILLIAM F. HEISLER, Colonel, USAF
Chief, Command Systems Division
Directorate of Planning and Technology

## ABSTRACT

MI-3 is a primitive, interactive, one pass assembler which assembles in-core code for a family of microprogrammed computers based on an Interdata 3 (I-3) micromachine.

TABLE OF CONTENTS

TABLE OF CONTENTS (Concluded)

# SECTION I

## INTRODUCTION

MI-3 is a primitive, interactive, one pass assembler which assembles in-core code for a family of microprogrammed computers based on an Interdata 3 (I-3) micromachine.

Versions of MI-3 have been assembled and operate under the Calliope and Venus microprograms. Calliope is a MITRE-produced superset of the I-3 delivered machine. Venus, also MITRE produced, provides multiprogramming capabilities; it includes most instructions implemented in Calliope, plus many others.

MI-3 was written for interim use, to allow time for a more powerful, flexible and useful assembler to be built. As a result, it betrays a number of anomalies and idiosyncracies not normally to be expected of a more finished product. These are described in Section VII.

## SECTION II

## DATA TYPES

### INTEGERS

All integers are represented in the MI-3 Assembly languages in hex (base 16) as a string of hex digits. No facilities are provided for representing integers in either decimal or binary.

### SYMBOLS

Three kinds of symbols are implemented in MI-3, known as S, G and L symbols.

#### S Symbols

An S symbol is formed by writing S followed by two hex digits. These symbols are unusual, in that they may be defined as often as desired. When S symbols are referenced, a B or F must be appended, specifying whether the symbol is defined before (Backward) or ahead (Forward) of the reference.

The occurrence of a backward S symbol in a line of code refers to the closest previous definition of that symbol. The occurrence of a forward S symbol in a line of code refers to the closest following definition of that symbol. S symbol references never refer to the line of code in which they occur.

Thus, for example,

```
SØ1     LHI   R6,SØ1F
          .
          .
          .
SØ1     BAL   R7,SØ1B
```

The SØ1F in the LHI refers to the SØ1 on the BAL. The SØ1B on the BAL refers to the SØ1 on the LHI.

## G and L Symbols

G and L symbols consist of the letter G or L followed by 1 to 5 characters chosen from the alphabet and the digits.

For example:

```
LSA4
G123LM
LPQRST
```

Usage of G and L symbols is presently identical. An unimplemented addition would limit the scope (the set of statements over which they are defined) of L symbols so that they become, in a sense, Local. G symbols would not be so limited, and thus would be Global.

## REGISTER IDENTIFIERS

A programmer may use a register identifier when he wishes to draw special attention to the fact that a value is to designate a general purpose register (otherwise, an integer will do just as well).

A register identifer is written as the letter R followed by a single hex digit identifying the particular register.

Thus, R6 can be used instead of 6, RD instead of D.

## ASTERISK (*)

The * is a symbol which may be used to denote the current value of the location counter. When used in an instruction, its value is the address of the first byte of the instruction, in a DC or DS the address of the first byte assigned.

## SECTION III

## FORMATS

A program consists of a sequence of lines. Each line contains a LOC (location) field, an OP (operation code) field, a DATA field, and a COMMENTS field. Fields are separated from other fields by one or more spaces:

LOC OP DATA COMMENTS

Embedded spaces may not appear in the LOC, OP or DATA fields.

### LOC FIELD

The LOC field is optional on instructions and on the DC and DS pseudo-operations, required on an EQU line, and ignored on all others.

When present, it must start in the first input column, and consist of a G, L or S symbol. In the EQU line, the symbol in the LOC field is assigned the value of the operand of the EQU; in all other cases, the symbol takes on the value of the current location counter (*).

### OP FIELD

The OP field contains the name of an instruction or a pseudo-operation. It may be from 1 to 4 alphanumeric characters.

If the LOC field is absent, the OP field may still not start before the second input column. It is terminated by the first blank character.

### DATA FIELD

The DATA field, separated from the OP field by one or more spaces, contains the operands for instructions and pseudo-operations.

### COMMENTS FIELD

Input columns beyond the DATA field may be used to enter comments into the program, and may contain embedded blanks. The COMMENTS field is separated from the OP field by one or more spaces.

## SECTION IV

## EXPRESSIONS

The DATA field of all instructions and most pseudo-operations contain expressions. An expression is one or more symbols and/or numbers, connected with the operators + and/or - .

Arithmetic may be performed on any combinations of:

backward S symbols
register identifiers
integers
* (the current location counter)

Arithmetic may not be performed on:

forward S symbols
L symbols
G symbols

When these symbols are used, they must stand alone.

Each component of an expression is considered to be a 16-bit value; 16-bit sums and differences are computed in two's complement arithmetic. When the value of an expression is inserted into a field of an instruction and the field is less than 16 bits wide, the leftmost bits of the value are stripped off.

Examples:

LXYZ
GBQ3F
SAAF
S22B
*
2A
R5
SØ1B-7+*
RB+23
Ø-12

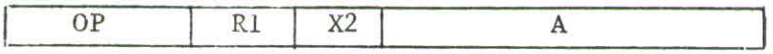
Counter Examples:

| | |
|---|---|
| L9FC+5<br>*-GRAB<br>SAAF-SØ1B | (Arithmetic not legal on L symbols, G symbols and forward S symbols) |
| -12 | ( + and - must appear between two symbols or numbers) |

## SECTION V

## INSTRUCTIONS

Four assembler formats, which assemble into two basic machine formats, are supported for instructions. The machine formats are either 16 or 32 bits long. The first 8 bits of each contains the operation code:


R1, R2 and X2 are 4-bit fields; A is 16 bits.

| Type | Data Field Format | Assembled Bits |
|---|---|---|
| 1 | R1,R2 | 16 |
| 2 | R1,A(X2) or R1,A | 32 |
| 3 | R2 | 16 |
| 4 | A(X2) or A | 32 |

Any expression may be used to define the A field. The R1, R2 and X2 fields may be defined using expressions, but not L, G or forward S symbols. Expressions used to define the R1, R2 and X2 fields are evaluated as 16 bits, then truncated to 4 bits.

Except for the always optional X2 field, fields may not be omitted; Ø may be entered instead.

In operations of types 1 and 2, the R1 field usually refers to one of the general registers. In the BTC, BTCR, BFC and BFCR instructions, however, the R1 field is a mask which determines the conditions to be tested. Type 3 and 4 operations represent extended mnemonics for such instructions in which the mask, i.e. the R1 field, is implicit in the mnemonic.

OPERATION CODES OF TYPE 1

An operation code of type 1 requires two operands, R1 and R2, both four bits in length. It is written in the form:

```
LOC     OP     R1,R2
```

and is assembled into 16 bits:


Examples:

```
LBR   R6,R5
BALR  SØ1B+3,*-S3FB
```

Counter Examples:

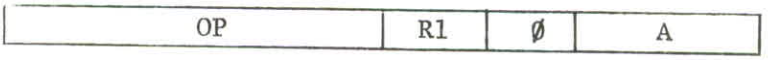
```
BTCR  LABC,R7      (L and G symbols may not
MHR   R5,G124       be used to define an R1
                    or R2 field)

STBR  ,RC          (operand may not be omitted)
```

OPERATION CODES OF TYPE 2

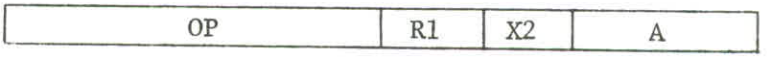
An operation code of type 2 requires two operands, R1 (4 bits) and A or A(X2). R1 and X2 are 4 bits long, and A is 16 bits. It is written:

```
LOC     OP      R1,A

        or

LOC     OP      R1,A(X2)
```

and is assembled into 32 bits:




or




respectively.

Examples:

```
AHI     R6,24
NH      X2,GA(R5)
BAL     RF,LABEL
XHI     RA,LOC(R1+R2-SØ1B)
```

Counter Examples:

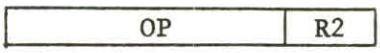
```
STBS    3,LX(GAX)       (G symbol not allowed
                         in X2 field)

WD      R4              (missing operand)
```

OPERATION CODES OF TYPE 3

An operation code of type 3 requires one operand, R2, 4 bits long. It is written in the form:

```
LOC     OP     R2
```

and assembles as 16 bits:




Operations of this type represent extended mnemonics for the instructions BTCR and BFCR, with R1 set implicitly. Those defined are listed in Appendix A.

Example:

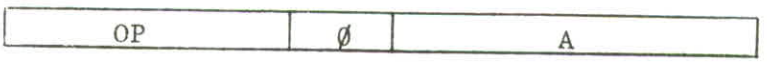
```
BR     RF
```

OPERATION CODES OF TYPE 4

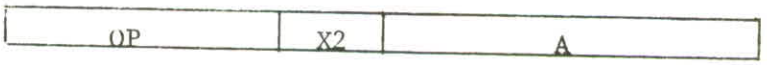
An operation code of type 4 requires one operand, A or A(X2), where A is 16 bits long, and X2 is 4 bits. It is written:

```
LOC     OP     A

        or

LOC     OP     A(X2)
```

and assembles into 32 bits:




or




respectively.

Operations of this type are extended mnemonics for the BTC and BFC basic instructions, using implicit values for the R1 field. Those defined are listed in Appendix A.

Examples:

```
BZ      LABEL          (same as BFC  3,LABEL)

BO      SØ1B-5(R6)     (same as BTC
                        4,SØ1B-5(R6))
```

## SECTION VI

## PSEUDO-OPERATIONS

### DC (DEFINE CONSTANT)

A DC line is used to define a single, 16-bit constant:

```
LOC     DC     EXPRESSION
```

### DS (DEFINE STORAGE)

A DS line is used to reserve a number of bytes in storage:

```
LOC     DS     EXPRESSION
```

Symbols used in the expression must have been previously defined.

The line

```
SØØ     DS     35C
```

is equivalent to

```
SØØ     EQU    *
        ORG    *+35C
```

END

An END line is used to indicate the end of the source program and the address of the first memory location to be executed.

```
        END    OPERAND
```

OPERAND may be any expression, but it must be present and defined (it may be Ø).

Examples:

```
        END    LSTART
        END    Ø
```

Counter Examples:

```
        END                (operand must be present)
        END    SA5F        (operand undefined)
```

EQU (EQUALS)

An EQU line is used to define a symbol without generating a line of object code:

```
LOC     EQU    EXPRESSION
```

where LOC is a G, L or S symbol. Symbols used in the expression must have been previously defined.

OPD (OPERATION DEFINITION)

The OPD line may be used to add instruction and pseudo-operation mnemonics to the set recognized by MI-3. In fact, MI-3 itself uses the OPD to define all mnemonics except OPD itself, which is built in. When MI-3 is assembled, a deck of OPD cards is also assembled. For the Venus instruction set, the OPD cards are listed in Appendixes C and D.

OPD is useful for defining new mnemonics for instructions, or synonyms for pseudo-operations. Only additions are possible; old mnemonics may not be deleted, nor may values associated with them be changed, or entirely new pseudo-operations added.

OPD is written:

```
OPD    'OPNAME',OPCODE,FORMAT
```

which is assembled into the operation table as:

| OPNAME | OPCODE | FORMAT |
|---|---|---|

where

OPNAME is a 1 to 4-character alphanumeric string which is the desired mnemonic.

OPCODE is an expression in which all symbols have been previously defined. It specifies the 8 bit operation code to be associated with the name OPNAME for instructions. It must be zero for pseudo-operations.

FORMAT is an expression in which all symbols have been previously defined. It is an 8-bit field; the last 4 bits specify the instruction types 1, 2, 3 or 4 or the pseudo-operation type (see Appendix A).

The first 4 bits of FORMAT are meaningless except for instruction types 3 and 4, in which they specify the value to be used in the R1 field.

Examples:

```
OPD   'STBR',22,Ø1      (type 1, OP = 22)

OPD   'BL',72,84        (type 4, OP = 72,
                         R1 = 8)

OPD   'OP',ØØ,Ø8        (type 8, OP = Ø for
                         all pseudo-operations)
                        (adds a new name for
                         DC)
```

Note that since definitions are made directly into core, a definition is permanent until a new copy of the assembler is loaded.

ORG (ORIGIN)

An ORG line is used to set the value of the location counter:

```
ORG   EXPRESSION
```

where symbols used in the expression must have been previously defined.

PUT

A PUT line is used when a program is to be assembled into locations other than those from which it will be executed. Thus, MI-3 must assemble the instructions as though they were in the locations from which they will be executed, but it must PUT them in a different place.

The PUT is used most often in one of two situations: to relocate code which will later overlay part of the assembler; and to assemble code with origin zero (for relocation by index) without destroying the low address region of memory.

The PUT is written:

```
PUT   EXPRESSION
```

where symbols used in the expression must have been previously defined.

The effect of a PUT line is to define a constant that is added to the assembled address of each line of code to obtain the address in core at which it will be placed. The addition is done modulo $2^{16}$, causing a wrap-around effect.

Example 1:

```
            ORG   25ØØ
            PUT   1ØØØ
SØ1         B     *
```

SØ1 and * will be defined as 25ØØ, and the instruction will be stored in 35ØØ.

Example 2:

```
            ORG   55ØØ
            PUT   EDØØ
            B     *
```

* is assigned the value 55ØØ, and the instruction stored at location 42ØØ, since EDØØ is equivalent to -13ØØ. It could also have been written:

```
            PUT   Ø-13ØØ
```

Example 3:

```
            ORG   47E2
S11         EQU   *
            PUT   45ØØ-S11B
            B     *
```

assembles into location 45ØØ, with * and S11 defined as 47E2.

Note that in Example 3 the output of the assembly will be placed at 1ØØØ regardless of where it is ORGed.

SID (SET INPUT DEVICE)

A SID line controls the source of symbolic input to MI-3. The only valid possibilities are:

| | | |
|---|---|---|
| SID | 0 | paper tape |
| SID | 1 | keyboard |
| SID | 2 | card reader (initial value) |

The operand field may not contain an expression; only the explicit values, Ø, 1 and 2 may be used.

SOD (SET OUTPUT DEVICE)

A SOD line controls the printing of a listing and the output device on which it is to be printed. The only possibilities are:

| | | |
|---|---|---|
| SOD | 1 | teletype |
| SOD | 2 | printer (initial value) |
| SOD | 3 | no print |

The operand field may not contain an expression; only the explicit values 1, 2 and 3 may be used.

## SECTION VII

## IDIOSYNCRASIES

Some of the idiosyncrasies of MI-3 are discussed briefly below. It is possible that as time goes on changes may be made to MI-3 which eliminate or change some of them.

### MESSAGES

There are only four messages built into MI-3. All four are forced to the operator's teletype. Two of them announce the beginning and ending of an assembly. The third recognizes a system failure which has occurred in the form of an illegal instruction during the assembly. The fourth, consisting of the words ERROR IN FOLLOWING LINE, represents MI-3's total capability for diagnosing user errors. At this point, the user must enter a valid line on the teletype which is to replace the one found to be in error.

### INPUT

When using the on-line teletype as an input device, MI-3 provides no editing capabilities whatsoever. The usual abilities to cancel a line or to backspace characters are not present.

The specifications state that the end of the DATA field is determined by a space. In fact, it is determined by the first character which MI-3 recognizes as being invalid in a data field. Thus, for example, characters such as & or % will, in fact, terminate the data field without any error indication.

When MI-3 is first entered, it identifies itself by logging a message on the console teletype. If the card reader, the assumed input device, is not ready with cards, MI-3 will wait until it is, with no indication of what it is waiting for. Similarly, MI-3 will wait until the printer, the assumed listing output device, is ready; it will have read the first card. In both cases, readying the device allows operation of MI-3 to proceed without restarting.

OUTPUT

The binary results of the assembly are stored directly into core memory. There is no other computer processable output.

The output listing does not print out the binary values assembled nor their locations in memory. While S symbol values are logged when the S symbol is defined, no indication is given of the value of G or L symbols. Undefined symbols are not listed, nor, for that matter, are defined symbols.

LANGUAGE

The concept of S symbols appears entirely unique to MI-3. The fact that a particular S symbol can appear any number of times in a given assembly is, perhaps, the most unusual aspect. Coupled with this is the need for the programmer to indicate to MI-3 whether the particular S symbol referred to has been defined earlier or will be defined later in the assembly.

Character strings may not be written explicitly. The only way character strings can be specified is to write tham as the series of equivalent hex constants in DC statements, or in hex as instruction operands.

All numbers entered by a programmer as part of the program must be entered in hex. No provision is made to handle numbers written in decimal (base 10) form.

## SECTION VIII

## EXAMPLES

In the examples below, the output of the assembler is shown as a 4 hex digit location, followed by a colon, followed by 4 or 8 hex digits representing the code stored in that location.

Example 1:

```
        ORG   2ØØØ
        B     4(3)
        XHR   R1,RD
        LHI   RC,3DE1
        BR    FC
```

assembles into:

```
2ØØØ:   74Ø3  ØØØ4
2ØØ4:   C71D
2ØØ6:   D8CØ  3DE1
2ØØA:   84ØC
```

Example 2:

```
        ORG   1FØØ
        B     *
        B     *+*
        B     *-*+1
        B     4-1-1-1-1
```

assembles into:

```
1FØØ:   74ØØ  1FØØ
1FØ4:   74ØØ  3EØ8
1FØ8:   74ØØ  ØØØ1
1FØC:   74ØØ  ØØØØ
```

Example 3:

```
                ORG     3ØØØ
SØØ             B       SØØF
                B       SØØB
SØØ             B       SØØB
                B       SFCF
                B       SØØB
SFC             B       SØØB
```

assembles into:

```
3ØØØ:   74ØØ    3Ø08
3ØØ4:   74ØØ    3ØØØ
3ØØ8:   74ØØ    3ØØØ
3ØØC:   74ØØ    3Ø14
3Ø1Ø:   74ØØ    3Ø08
3Ø14:   74ØØ    3Ø08
```

Example 4:

```
                ORG     4ØFØ
                DC      897FD1C8B
SØ8             DC      SØ7F
                DC      SØ8B-1
                DC      *
SØ7             DC      SØ8B+*
```

assembles into:

```
4ØFØ:   1C8B
4ØF2:   4ØF8
4ØF4:   4ØF1
4ØF6:   4ØF6
4ØF8:   81EA
```

Example 5:

```
                ORG     44EØ
SØ9             DS      2ØØ
S1Ø             DC      SØ9B
S11             DS      4
                B       *
```

assembles into:

```
46EØ:   44EØ
46E6:   74ØØ  46E6
```

## APPENDIX I

## MNEMONICS AND VALUES

The set of mnemonics supplied in the version of MI-3 which runs on the Venus machine is summarized below. As far as the assembler is concerned, mnemonics fall into fifteen types, each identified by a type code. The type determines the format and meaning of the DATA field of instructions, and identifies extended mnemonics and pseudo-operations:

| | | |
|---|---|---|
| 1 | | 16 bit instructions |
| 2 | | 32 bit instructions |
| 3 | | Extended mnemonics for 16 bit instructions (BTCR, BFCR) |
| 4 | | Extended mnemonics for 32 bit instructions (BTC, BFC) |
| 5 | ORG | Origin |
| 6 | EQU | Equals |
| 7 | END | End |
| 8 | DC | Define Constant |
| 9 | DS | Define Storage |
| A | PUT | Put |
| B | OPD | Operation Definition |
| C | SOD | Set Output Device |
| D | SID | Set Input Device |
| E | LEND | Local End (Not implemented) |
| F | | Not assigned |

Instructions appear in a 16 x 16 matrix in which the 8-bit operation code is formed by taking the row number in hex followed by the column number, also in hex. For example, CALL is in row 7, column A; its operation code is 7A. Instructions in rows 2, 8 and C are of type 1; those in the other rows, type 2.

Extended operations are listed separately.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | | |
| 1 | P | V | STB | POB | STH | PO | SSN | | | | | OC | RD | WD | SS | |
| 2 | | | STBR | POBR | STHR | POR | SSNR | | | | | OCR | RDR | WDR | SSR | JOBA |
| 3 | PS | VS | STBS | POBS | STHS | POS | SSNS | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | |
| 5 | DIE | UNQP | PUC | POC | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | |
| 7 | BXLE | BAL | BTC | BXH | BFC | SIO | EL1 | SET | RSET | RETN | CALL | ICOR | SRHL | SLHL | SRHA | RLH |
| 8 | BXLR | BALR | BTCR | BXHR | BFCR | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | |
| A | | | | | | | | | | | | | | | | |
| B | | PU | PUB | LSN | NH | CLH | OH | XH | LH | LB | AH | SH | MH | DH | ACH | SCH |
| C | | PUR | PUBR | LSNR | NHR | CLHR | OHR | XHR | LHR | LBR | AHR | SHR | MHR | DHR | ACHR | SCHR |
| D | | PUI | PUBI | LSNI | NHI | CLHI | OHI | XHI | LHI | LBI | AHI | SHI | MHI | DHI | ACHI | SCHI |
| E | | PUS | PUBS | LSNP | NHS | CLHS | OHS | XHS | LHS | LBS | AHS | SHS | MHS | DHS | ACHS | SCHS |
| F | | PUP | PUBP | LSNS | NHP | CLHP | OHP | XHP | LHP | LBP | AHP | SHP | MHP | DHP | ACHP | SCHP |

EXTENDED OPERATIONS

| Hex | Mnemonic | Equivalent Operation | Meaning |
|---|---|---|---|
| 720 | NOP | BTC Ø | No operation |
| 721 | BM | BTC 1 | Branch on minus |
| 722 | BP | BTC 2 | Branch on plus |
| 723 | BNZ | BTC 3 | Branch on non-zero |
| 723 | BNE | BTC 3 | Branch on not equal |
| 724 | BO | BTC 4 | Branch on overflow |
| 728 | BC | BTC 8 | Branch on carry |
| 728 | BL | BTC 8 | Branch on low |
| 740 | B | BFC Ø | Branch |
| 741 | BNM | BFC 1 | Branch on non-minus |
| 742 | BNP | BFC 2 | Branch on non-plus |
| 743 | BZ | BFC 3 | Branch on zero |
| 743 | BE | BFC 3 | Branch on equal |
| 748 | BNC | BFC 8 | Branch on no carry |
| 748 | BNL | BFC 8 | Branch on not low |
| 820 | NOPR | BTCR Ø | No operation |
| 820 | BR | BFCR Ø | Branch |

In the hex equivalent, the first two digits are the operation code of the basic instruction; the last digit is the R1 field. The operations based on BFC and BTC are type 4; those based on BTCR or BFCR are of type 3.

## APPENDIX II

## FORMAL SYNTAX

### BASIC DEFINITIONS

A lower case b is used to denote a single blank character.

<space> :: = b | <space> b

<hex digit> :: = Ø|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F

<integer> :: = <hex digit> | <integer> <hex digit>

<S symbol> :: = S <hex digit> <hex digit>

<backward local reference> :: = <S symbol> B

<forward local reference> :: = <S symbol> F

<alphabetic> :: = A|B|...|Z

<digit> :: = Ø|1|2|3|4|5|6|7|8|9

<alphanumeric> :: = <alphabetic> | <numeric>

<L symbol> :: = L <up to 5 alphanumeric>

<G symbol> :: = G <up to 5 alphanumeric>

<symbol> :: = <S symbol> | <L symbol> | <G symbol>

<loc field> :: = <symbol> | <empty>

<register identifier> :: = R <hex digit>

### EXPRESSIONS

<expression> :: = <forward local reference> |
    <proper expression> | <L symbol> | <G symbol>

<defined value> :: = <proper expression> |
    <previously defined L symbol> |
    <previously defined G symbol>

<additive operator> :: = + | -

<term> :: = <integer> | <backward local reference> |
    <register identifier> | *

<proper expression> :: = <term> | <proper expression>
    <additive operator> <term>

LINES

<line> :: = <loc field> <space> <basic line> | <ORG line> |
<END line> | <DC line> | <DS line> | <EQU line> |
<OPD line> | <PUT line> | <SID line> | <SOD line>

<basic line> :: = <op code of type 1> <space> <data field of type 1> |
<op code of type 2> <space> <data field of type 2> |
<op code of type 3> <space> <data field of type 3> |
<op code of type 4> <space> <data field of type 4>

<data field of type 1> :: = <4 bit operand>,<4 bit operand>

<data field of type 2> :: = <4 bit operand>,<16 bit operand> |
<4 bit operand>,<16 bit operand> (<4 bit operand>)

<data field of type 3> :: = <4 bit operand>

<data field of type 4> :: = <16 bit operand> |
<16 bit operand> (<4 bit operand>)

<4 bit operand> :: = <proper expression>

<16 bit operand> :: = <expression>

<DC line> :: = <loc field> <space> DC <space> <expression>

<DS line> :: = <loc field> <space> DS <space> <defined value>

<END line> :: = <space> END <space> <defined value>

<EQU line> :: = <symbol> <space> EQU <space> <defined value>

<OPD line> :: = <space> OPD <space> '<up tp 4 alphanumeric>',
<defined value>,<defined value>

<ORG line> :: = <space> ORG <space> <defined value>

<PUT line> :: = <space> PUT <space> <proper expression>

<SID line> :: = <space> SID <space> <input device>

<input device> :: = 0|1|2

<SOD line> :: = <space> SOD <space> <output control>

<output control> :: = 1|2|3

## APPENDIX III

## ALPHABETIC LIST OF MNEMONICS

```
OPD 'ACH',BE,02          ADD WITH CARRY HALFWORD
OPD 'ACHI',DE,02         ADD WITH CARRY HALFWORD
OPD 'ACHP',FE,02         ADD WITH CARRY HALFWORD
OPD 'ACHR',CE,01         ADD WITH CARRY HALFWORD
OPD 'ACHS',EE,02         ADD WITH CARRY HALFWORD
OPD 'AH',BA,02           ADD HALFWORD
OPD 'AHI',DA,02          ADD HALFWORD.
OPD 'AHP',FA,02          ADD  HALFWORD
OPD 'AHR',CA,01          ADD HALFWORD
OPD 'AHS',EA,02          ADD HALFWORD
OPD 'B',74,04            BRANCH UNCONDITIONAL
OPD 'BAL',71,02          BRANCH AND LINK
OPD 'BALR',81,01         BRANCH AND LINK
OPD 'BC',72,84           BRANCH ON CARRY
OPD 'BE',74,34           BRANCH ON EQUAL
OPD 'BER',84,01          BRANCH EQUAL REGISTER
OPD 'BFC',74,02          BRANCH ON FALSE CONDITION
OPD 'BFCR',84,01         BRANCH ON FALSE CONDITION
OPD 'BL',72,84           BRANCH ON LOW
OPD 'BM',72,14           BRANCH ON MINUS
OPD 'BNC',74,84          BRANCH ON NO CARRY
OPD 'BNE',72,34          BRANCH ON NOT EQUAL
OPD 'BNL',74,84          BRANCH ON NOT LOW
OPD 'BNM',74,24          BRANCH ON NOT MINUS
OPD 'BNP',74,24          BRANCH ON NOT PLUS
OPD 'BNZ',72,34          BRANCH ON NOT ZERO
OPD 'BO',72,44           BRANCH ON OVERFLOW
OPD 'BP',72,24           BRANCH ON PLUS
OPD 'BR',84,03           BRANCH UNCONDITIONAL
OPD 'BTC',72,02          BRANCH ON TRUE CONDITION
OPD 'BTCR',82,01         BRANCH ON TRUE CONDITION
OPD 'BXH',73,02          BRANCH ON INDEX HIGH
OPD 'BXHR',83,01         BRANCH ON INDEX HIGH
OPD 'BXLE',70,02         BRANCH ON INDEX LOW OR EQUAL
OPD 'BXLR',80,01         BRANCH ON INDEX LOW OR EQUAL
OPD 'BZ',74,34           BRANCH ON ZERO
OPD 'CALL',7A,02         SUBROUTINE CALL
OPD 'CLH',B5,02          COMPARE LOGICAL HALFWORD
OPD 'CLHI',D5,02         COMPARE LOGICAL HALFWORD
OPD 'CLHP',FE,02         COMPARE LOGICAL HALFWORD
OPD 'CLHR',C5,01         COMPARE LOGICAL HALFWORD
OPD 'CLHS',E5,02         COMPARE LOGICAL HALFWORD
OPD 'DC',00,08           DEFINE CON
OPD 'DH',BD,02           DIVID HALFWORD
```

```
OPD 'DHI',DD,02          DIVIDE HALFWORD
OPD 'DHP',FD,02          DIVIDE HALFWORD
OPD 'DHR',CD,01          DIVIDE HALFWORD
OPD 'DHS',ED,02          DIVIDE HALFWORD
OPD 'DIE',60,01          JOB SUICIDE
OPD 'DS',00,09           DEFINE STORAGE
OPD 'EL1',76,02          ENTER LEVEL 1
OPD 'END',00,07          END
OPD 'EQU',00,06          EQUAL
OPD 'ICOR',7B,02         CHECK FOR PAGE IN CORE
OPD 'JOBA',2F,01         FETCH JOB AREA LOCATION
OPD 'LB',B9,02           LOAD BYTE
OPD 'LBI',D9,02          LOAD BYTE
OPD 'LBP',F9,02          LOAD BYTE
OPD 'LBR',C9,01          LOAD BYTE
OPD 'LBS',E9,02          LOAD BYTE
OPD 'LEND',00,0E         LOCAL END
OPD 'LH',B8,02           LOAD HALFWORD
OPD 'LHI',D8,02          LOAD HALFWORD IMMEDIATE
OPD 'LHP',F8,02          LOAD HALFWORD
OPD 'LHR',C8,01          LOAD HALFWORD
OPD 'LHS',E8,02          LOAD HALFWORD
OPD 'LSN',B3,02          LOAD STREAM NAME
OPD 'LSNI',D3,02         LOCAL STREAM NAME
OPD 'LSNP',F3,02         LOCAL STREAM NAME
OPD 'LSNR',C3,01         LOCAL STREAM NAME
OPD 'LSNS',E3,02         LOCAL STREAM NAME
OPD 'MH',BC,02           MULTIPLY HALFWORD
OPD 'MHI',DC,02          MULTIPLY HALFWORD
OPD 'MHP',FC,02          MULTIPLY HALFWORD
OPD 'MHR',CC,01          MULTIPLY HALFWORD
OPD 'MHS',EC,02          MULTIPLY HALFWORD
OPD 'NH',B4,02           AND HALFWORD
OPD 'NHI',D4,02          AND HALFWORD
OPD 'NHP',F4,02          AND HALFWORD
OPD 'NHR',C4,01          AND HALFWORD
OPD 'NHS',E4,02          AND HALFWORD
OPD 'NOP',72,04          NO OPERATION
OPD 'NOPR',82,03         NO OPERATION
OPD 'OC',1B,02           OUTPUT COMMAND
OPD 'OCP',2B,01          OUTPUT COMMAND
OPD 'OH',B6,02           OR HALFWORD
OPD 'OHI',D6,02          OR HALFWORD
OPD 'OHP',F6,02          OR HALFWORD
```

```
OPD 'OHR',C6,01          OR HALFWORD
OPD 'OHS',F6,02          OR HALFWORD
OPD 'ORG',00,0F          ORIGIN
OPD 'P',10,02            P OF SEMAPHORE
OPD 'PO',15,02           POP FROM STACK
OPD 'POB',13,02          POP BYTE FROM STACK
OPD 'POBD',23,01         POP BYTE FROM STACK
OPD 'POBS',33,02         POP BYTE FROM STACK
OPD 'POC',53,01          POP FROM CONTROL STACK
OPD 'POD',25,01          POP FROM STACK
OPD 'POS',35,02          POP FROM STACK
OPD 'PS',30,02           P OF SEMAPHORE
OPD 'PU',B1,02           PUSH HALFWORD INTO STACK
OPD 'PUB',B3,02          PUSH BYTE INTO STACK
OPD 'PUBI',D3,02         PUSH BYTE INTO STACK
OPD 'PUBP',F3,02         PUSH BYTE INTO STACK
OPD 'PUBR',C3,01         PUSH BYTE INTO STACK
OPD 'PUBS',F3,02         PUSH BYTE INTO STACK
OPD 'PUC',53,01          PUSH INTO CONTROL STACK
OPD 'PUI',D1,02          PUSH HALFWORD INTO STACK
OPD 'PUP',F1,02          PUSH FROM PROGRAM
OPD 'PUR',C1,01          PUSH HALFWORD INTO STACK
OPD 'PUS',F1,02          PUSH HALFWORD INTO STACK
OPD 'PUT',00,0A          PUT
OPD 'RD',1C,02           READ DATA
OPD 'RDR',3C,01          READ DATA
OPD 'RETN',79,02         SUBROUTINE RETURN
OPD 'RLH',7F,02          ROTATE LEFTWARD HALFWORD
OPD 'RSET',78,02         RESET CONDITION/ON REGISTER
OPD 'SCH',BF,02          SUBTRACT WITH CARRY HALFWORD
OPD 'SCHI',DF,02         SUBTRACT WITH CARRY HALFWORD
OPD 'SCHP',FF,02         SUBTRACT WITH CARRY HALFWORD
OPD 'SCHR',CF,01         SUBTRACT WITH CARRY HALFWORD
OPD 'SCHS',EF,02         SUBTRACT WITH CARRY HALFWORD
OPD 'SET',77,02          SET CONDITION/ON REGISTER
OPD 'SH',BB,02           SUBTRACT HALFWORD
OPD 'SHI',DB,02          SUBTRACT HALFWORD
OPD 'SHP',FB,02          SUBTRACT HALFWORD
OPD 'SHR',CB,01          SUBTRACT HALFWORD
OPD 'SHS',EB,02          SUBTRACT HALFWORD
OPD 'SID',00,0D          SYSTEM INPUT DEVICE
OPD 'SIO',75,02          START I/O CHANNEL
OPD 'SLHL',7D,02         SHIFT LEFT LOGICAL
OPD 'SOD',00,0C          SYSTEM OUTPUT DEVICE
```

```
OPD 'SRHA',7E,02        SHIFT RIGHT ARITHMETIC
OPD 'SRHL',7C,02        SHIFT RIGHT LOGICAL
OPD 'SS',1F,02          SENSE STATUS
OPD 'SSN',16,02         STORE STREAM NAME
OPD 'SSNR',26,01        STORE STREAM NAME
OPD 'SSNS',36,02        STORE STREAM NAME
OPD 'SSR',2F,01         SENSE STATUS
OPD 'STB',12,02         STORE BYTE
OPD 'STBR',22,01        STORE BYTE
OPD 'STBS',32,02        STORE BYTE
OPD 'STH',14,02         STORE HALFWORD
OPD 'STHR',24,01        STORE HALFWORD
OPD 'STHS',34,02        STORE HALFWORD
OPD 'UNQR',51,01        UNQUEUE WHEN DISK SWAP COMPLETE
OPD 'V',11,02           V OF SEMAPHORE
OPD 'VS',31,02          V OF SEMAPHORE
OPD 'WD',1D,02          WRITE DATA
OPD 'WDR',2D,01         WRITE DATA
OPD 'XH',B7,02          EXCLUSIVE OR HALFWORD
OPD 'XHI',D7,02         EXCLUSIVE OR HALFWORD IMMEDIATE
OPD 'XHR',F7,02         EXCLUSIVE OR HALFWORD
OPD 'XHR',C7,01         EXCLUSIVE OR HALFWORD
OPD 'XHS',F7,02         EXCLUSIVE OR HALFWORD
```

## APPENDIX IV

## NUMERIC LIST OF MNEMONICS

```
OPD 'ORG',00,05          ORIGIN
OPD 'EQU',00,06          EQUAL
OPD 'END',00,07          END
OPD 'DC',00,08           DEFINE CON
OPD 'DS',00,09           DEFINE STORAGE
OPD 'PUT',00,0A          PUT
OPD 'SOD',00,0C          SYSTEM OUTPUT DEVICE
OPD 'SID',00,0D          SYSTEM INPUT DEVICE
OPD 'LEND',00,0F         LOCAL END
OPD 'P',10,02            P OF SEMAPHORE
OPD 'V',11,02            V OF SEMAPHORE
OPD 'STB',12,02          STORE BYTE
OPD 'POB',13,02          POP BYTE FROM STACK
OPD 'STH',14,02          STORE HALFWORD
OPD 'PO',15,02           POP FROM STACK
OPD 'SSN',16,02          STORE STREAM NAME
OPD 'OC',1B,02           OUTPUT COMMAND
OPD 'RD',1C,02           READ DATA
OPD 'WD',1D,02           WRITE DATA
OPD 'SS',1E,02           SENSE STATUS
OPD 'STBR',22,01         STORE BYTE
OPD 'POBR',23,01         POP BYTE FROM STACK
OPD 'STHR',24,01         STORE HALFWORD
OPD 'POR',25,01          POP FROM STACK
OPD 'SSNR',26,01         STORE STREAM NAME
OPD 'OCR',2B,01          OUTPUT COMMAND
OPD 'RDR',2C,01          READ DATA
OPD 'WDR',2D,01          WRITE DATA
OPD 'SSR',2E,01          SENSE STATUS
OPD 'JOBA',2F,01         FETCH JOB AREA LOCATION
OPD 'PS',30,02           P OF SEMAPHORE
OPD 'VS',31,02           V OF SEMAPHORE
OPD 'STBS',32,02         STORE BYTE
OPD 'POBS',33,02         POP BYTE FROM STACK
OPD 'STHS',34,02         STORE HALFWORD
OPD 'POS',35,02          POP FROM STACK
OPD 'SSNS',36,02         STORE STREAM NAME
OPD 'DIE',50,01          JOB SUICIDE
OPD 'UNQP',51,01         UNQUEUE WHEN DISK SWAP COMPLETE
OPD 'PUC',52,01          PUSH INTO CONTROL STACK
OPD 'POC',53,01          POP FROM CONTROL STACK
OPD 'BXLE',70,02         BRANCH ON INDEX LOW OR EQUAL
OPD 'BAL',71,02          BRANCH AND LINK
OPD 'BTC',72,02          BRANCH ON TRUE CONDITION
```

```
OPD  'NOP',72,04          NO OPERATION
OPD  'BM',72,14           BRANCH ON MINUS
OPD  'BP',72,24           BRANCH ON PLUS
OPD  'BNE',72,34          BRANCH ON NOT EQUAL
OPD  'BNZ',72,34          BRANCH ON NOT ZERO
OPD  'BO',72,44           BRANCH ON OVERFLOW
OPD  'BC',72,84           BRANCH ON CARRY
OPD  'BL',72,84           BRANCH ON LOW
OPD  'BXH',73,02          BRANCH ON INDEX HIGH
OPD  'BFC',74,02          BRANCH ON FALSE CONDITION
OPD  'B',74,04            BRANCH UNCONDITIONAL
OPD  'BNM',74,24          BRANCH ON NOT MINUS
OPD  'BNP',74,24          BRANCH ON NOT PLUS
OPD  'BE',74,34           BRANCH ON EQUAL
OPD  'BZ',74,34           BRANCH ON ZERO
OPD  'BNC',74,84          BRANCH ON NO CARRY
OPD  'BNL',74,84          BRANCH ON NOT LOW
OPD  'SIO',75,02          START I/O CHANNEL
OPD  'EL1',76,02          ENTER LEVEL 1
OPD  'SET',77,02          SET CONDITION/ON REGISTER
OPD  'RSET',78,02         RESET CONDITION/ON REGISTER
OPD  'RETN',79,02         SUBROUTINE RETURN
OPD  'CALL',7A,02         SUBROUTINE CALL
OPD  'ICOR',7B,02         CHECK FOR PAGE IN CORE
OPD  'SRHL',7C,02         SHIFT RIGHT LOGICAL
OPD  'SLHL',7D,02         SHIFT LEFT LOGICAL
OPD  'SRHA',7E,02         SHIFT RIGHT ARITHMETIC
OPD  'RLH',7F,02          ROTATE LEFTWARD HALFWORD
OPD  'BXLR',80,01         BRANCH ON INDEX LOW OR EQUAL
OPD  'BALR',81,01         BRANCH AND LINK
OPD  'BTCR',82,01         BRANCH ON TRUE CONDITION
OPD  'NOPR',82,03         NO OPERATION
OPD  'BXHR',83,01         BRANCH ON INDEX HIGH
OPD  'BER',84,01          BRANCH EQUAL REGISTER
OPD  'BFCR',84,01         BRANCH ON FALSE CONDITION
OPD  'BR',84,03           BRANCH UNCONDITIONAL
OPD  'PU',91,02           PUSH HALFWORD INTO STACK
OPD  'PUB',92,02          PUSH BYTE INTO STACK
OPD  'LSN',93,02          LOAD STREAM NAME
OPD  'NH',94,02           AND HALFWORD
OPD  'CLH',95,02          COMPARE LOGICAL HALFWORD
OPD  'OH',96,02           OR HALFWORD
OPD  'XH',97,02           EXCLUSIVE OR HALFWORD
OPD  'LH',98,02           LOAD HALFWORD
```

```
OPD 'LB',B9,02          LOAD BYTE
OPD 'AH',BA,02          ADD HALFWORD
OPD 'SH',BB,02          SUBTRACT HALFWORD
OPD 'MH',BC,02          MULTIPLY HALFWORD
OPD 'DH',BD,02          DIVID HALFWORD
OPD 'ACH',BE,02         ADD WITH CARRY HALFWORD
OPD 'SCH',BF,02         SUBTRACT WITH CARRY HALFWORD
OPD 'PUR',C1,01         PUSH HALFWORD INTO STACK
OPD 'PUBR',C2,01        PUSH BYTE INTO STACK
OPD 'LSNR',C3,01        LOCAL STREAM NAME
OPD 'NHR',C4,01         AND HALFWORD
OPD 'CLHR',C5,01        COMPARE LOGICAL HALFWORD
OPD 'OHR',C6,01         OR HALFWORD
OPD 'XHR',C7,01         EXCLUSIVE OR HALFWORD
OPD 'LHR',C8,01         LOAD HALFWORD
OPD 'LBR',C9,01         LOAD BYTE
OPD 'AHR',CA,01         ADD HALFWORD
OPD 'SHR',CB,01         SUBTRACT HALFWORD
OPD 'MHR',CC,01         MULTIPLY HALFWORD
OPD 'DHR',CD,01         DIVIDE HALFWORD
OPD 'ACHR',CE,01        ADD WITH CARRY HALFWORD
OPD 'SCHR',CF,01        SUBTRACT WITH CARRY HALFWORD
OPD 'PUI',D1,02         PUSH HALFWORD INTO STACK
OPD 'PUBI',D2,02        PUSH BYTE INTO STACK
OPD 'LSNI',D3,02        LOCAL STREAM NAME
OPD 'NHI',D4,02         AND HALFWORD
OPD 'CLHI',D5,02        COMPARE LOGICAL HALFWORD
OPD 'OHI',D6,02         OR HALFWORD
OPD 'XHI',D7,02         EXCLUSIVE OR HALFWORD IMMEDIATE
OPD 'LHI',D8,02         LOAD HALFWORD IMMEDIATE
OPD 'LBI',D9,02         LOAD BYTE
OPD 'AHI',DA,02         ADD HALFWORD
OPD 'SHI',DB,02         SUBTRACT HALFWORD
OPD 'MHI',DC,02         MULTIPLY HALFWORD
OPD 'DHI',DD,02         DIVIDE HALFWORD
OPD 'ACHI',DE,02        ADD WITH CARRY HALFWORD
OPD 'SCHI',DF,02        SUBTRACT WITH CARRY HALFWORD
OPD 'PUS',E1,02         PUSH HALFWORD INTO STACK
OPD 'PUBS',E2,02        PUSH BYTE INTO STACK
OPD 'LSNS',E3,02        LOCAL STREAM NAME
OPD 'NHS',E4,02         AND HALFWORD
OPD 'CLHS',E5,02        COMPARE LOGICAL HALFWORD
OPD 'OHS',E6,02         OR HALFWORD
OPD 'XHS',E7,02         EXCLUSIVE OR HALFWORD
```

```
OPD 'LHS',E8,02          LOAD HALFWORD
OPD 'LBS',E9,02          LOAD BYTE
OPD 'AHS',EA,02          ADD HALFWORD
OPD 'SHS',EB,02          SUBTRACT HALFWORD
OPD 'MHS',EC,02          MULTIPLY HALFWORD
OPD 'DHS',ED,02          DIVIDE HALFWORD
OPD 'ACHS',EE,02         ADD WITH CARRY HALFWORD
OPD 'SCHS',EE,02         SUBTRACT WITH CARRY HALFWORD
OPD 'PUP',F1,02          PUSH FROM PROGRAM
OPD 'PUBP',F2,02         PUSH BYTE INTO STACK
OPD 'LSNP',F3,02         LOCAL STREAM NAME
OPD 'NHP',F4,02          AND HALFWORD
OPD 'CLHP',F5,02         COMPARE LOGICAL HALFWORD
OPD 'OHP',F6,02          OR HALFWORD
OPD 'XHP',F7,02          EXCLUSIVE OR HALFWORD
OPD 'LHP',F8,02          LOAD HALFWORD
OPD 'LBP',F9,02          LOAD BYTE
OPD 'AHP',FA,02          ADD  HALFWORD
OPD 'SHP',FB,02          SUBTRACT HALFWORD
OPD 'MHP',FC,02          MULTIPLY HALFWORD
OPD 'DHP',FD,02          DIVIDE HALFWORD
OPD 'ACHP',FE,02         ADD WITH CARRY HALFWORD
OPD 'SCHP',FE,02         SUBTRACT WITH CARRY HALFWORD
END   0
```

APPENDIX V

KWIC INDEX LISTING

10/01/6-
05:52:2-

| KEYWORD AND TEXT | INDEX |
|---|---|
| ASTERISK ( * ) | 3 |
| ALPHABETIC LIST OF MNEMONICS | 31 |
| G AND L SYMBOLS | 3 |
| MNEMONICS AND VALUES | 25 |
| ASTERISK ( * ) | 3 |
| OPERATION CODES OF TYPE 1 | 9 |
| OPERATION CODES OF TYPE 2 | 10 |
| OPERATION CODES OF TYPE 3 | 11 |
| OPERATION CODES OF TYPE 4 | 12 |
| COMMENTS FIELD | 5 |
| DC ( DEFINE CONSTANT ) | 13 |
| DATA FIELD | 4 |
| DATA TYPES | 7 |
| DC ( DEFINE CONSTANT ) | 13 |
| DC ( DEFINE CONSTANT ) | 13 |
| DS ( DEFINE STORAGE ) | 13 |
| DPD ( OPERATION DEFINITION ) | 15 |
| SOD ( SET OUTPUT DEVICE ) | 18 |
| SID ( SET INPUT DEVICE ) | 18 |
| DS ( DEFINE STORAGE ) | 13 |
| END | 14 |
| EQU ( EQUALS ) | 14 |
| EQU ( EQUALS ) | 14 |
| EXAMPLES | 21 |
| EXPRESSIONS | 6 |
| COMMENTS FIELD | 5 |
| DATA FIELD | 4 |
| OP FIELD | 4 |
| LOC FIELD | 4 |
| FORMAL SYNTAX | 29 |
| FORMATS | 4 |
| G AND L SYMBOLS | 3 |

KWIC INDEX LISTING

10/01/69
05:52:26

| KEYWORD AND TEXT | INDEX |
|---|---|
| REGISTER IDENTIFIERS | 3 |
| IDIOSYNCRASIES | 19 |
| INDEX | 39 |
| INPUT | 19 |
| SID ( SET INPUT DEVICE ) | 18 |
| INSTRUCTIONS | 9 |
| INTEGERS | 2 |
| INTRODUCTION | 1 |
| G AND L SYMBOLS | 3 |
| LANGUAGE | 20 |
| ALPHABETIC LIST OF MNEMONICS | 31 |
| NUMERIC LIST OF MNEMONICS | 35 |
| LOC FIELD | 4 |
| MESSAGES | 19 |
| ALPHABETIC LIST OF MNEMONICS | 31 |
| NUMERIC LIST OF MNEMONICS | 35 |
| MNEMONICS AND VALUES | 2 |
| NUMERIC LIST OF MNEMONICS | 35 |
| ALPHABETIC LIST OF MNEMONICS | 31 |
| NUMERIC LIST OF MNEMONICS | 35 |
| OPERATION CODES OF TYPE 1 | 9 |
| OPERATION CODES OF TYPE 2 | 10 |
| OPERATION CODES OF TYPE 3 | 11 |
| OPERATION CODES OF TYPE 4 | 12 |
| OP FIELD | 4 |
| OPD ( OPERATION DEFINITION ) | 15 |
| OPERATION CODES OF TYPE 1 | 9 |
| OPERATION CODES OF TYPE 2 | 10 |
| OPERATION CODES OF TYPE 3 | 11 |
| OPERATION CODES OF TYPE 4 | 12 |
| OPD ( OPERATION DEFINITION ) | 15 |
| ORG ( ORIGIN ) | 16 |
| ORG ( ORIGIN ) | 16 |
| OUTPUT | 20 |
| SOD ( SET OUTPUT DEVICE ) | 19 |

KWIC INDEX LISTING

10/01/69
05:52:26

| KEYWORD AND TEXT | INDEX |
|---|---|
| PSEUDO-OPERATIONS | 13 |
| PUT | 16 |
| REGISTER IDENTIFIERS | 3 |
| S SYMBOLS | 2 |
| SID ( SET INPUT DEVICE ) | 18 |
| SOD ( SET OUTPUT DEVICE ) | 18 |
| SID ( SET INPUT DEVICE ) | 18 |
| SOD ( SET OUTPUT DEVICE ) | 18 |
| DS ( DEFINE STORAGE ) | 13 |
| S SYMBOLS | 2 |
| SYMBOLS | 2 |
| G AND L SYMBOLS | 3 |
| FORMAL SYNTAX | 29 |
| OPERATION CODES OF TYPE 1 | 9 |
| OPERATION CODES OF TYPE 2 | 10 |
| OPERATION CODES OF TYPE 3 | 11 |
| OPERATION CODES OF TYPE 4 | 12 |
| DATA TYPES | 2 |
| MNEMONICS AND VALUES | 25 |
| OPERATION CODES OF TYPE 1 | 9 |
| OPERATION CODES OF TYPE 2 | 10 |
| OPERATION CODES OF TYPE 3 | 11 |
| OPERATION CODES OF TYPE 4 | 12 |

UNCLASSIFIED
Security Classification

**DOCUMENT CONTROL DATA - R & D**

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

1. ORIGINATING ACTIVITY *(Corporate author)*
The MITRE Corporation
Bedford, Massachusetts

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
THE MI-3 ASSEMBLER REFERENCE MANUAL

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
N/A

5. AUTHOR(S) *(First name, middle initial, last name)*
R. W. Cornelli

| 6. REPORT DATE | 7a. TOTAL NO. OF PAGES | 7b. NO. OF REFS |
|---|---|---|
| DECEMBER 1969 | 46 | None |

| 8a. CONTRACT OR GRANT NO. | 9a. ORIGINATOR'S REPORT NUMBER(S) |
|---|---|
| F19(628)-68-C-0365 | |
| b. PROJECT NO. 700A | ESD-TR-69-371 |
| c. | 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* |
| d. | MTR-967 |

10. DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

| 11. SUPPLEMENTARY NOTES | 12. SPONSORING MILITARY ACTIVITY |
|---|---|
| N/A | Directorate of Planning and Technology, Electronic Systems Division, Air Force Systems Command, L. G. Hanscom Field, Bedford, Massachusetts |

13. ABSTRACT

MI-3 is a primitive, interactive, one pass assembler which assembles in-core code for a family of microprogrammed computers based on an Interdata 3 (I-3) micromachine.

DD FORM 1 NOV 65 1473

Security Classification

| 14. KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| MI-3 Assembler<br>Interdata 3<br>In-Core Code<br>Microprogrammed Computers | | | | | | |